RELATÓRIO ESINF - SPRINT 1

DIAGRAMA DE CLASSES E COMPLEXIDADE DE ALGORITMOS

GRUPO 21 – 2DC

1191507 – Bárbara Pinto

1200991 - Carlos Dias

1201029 - Cristóvão Sampaio

1201045 - Miguel Silva

1201154 – Martim Maciel

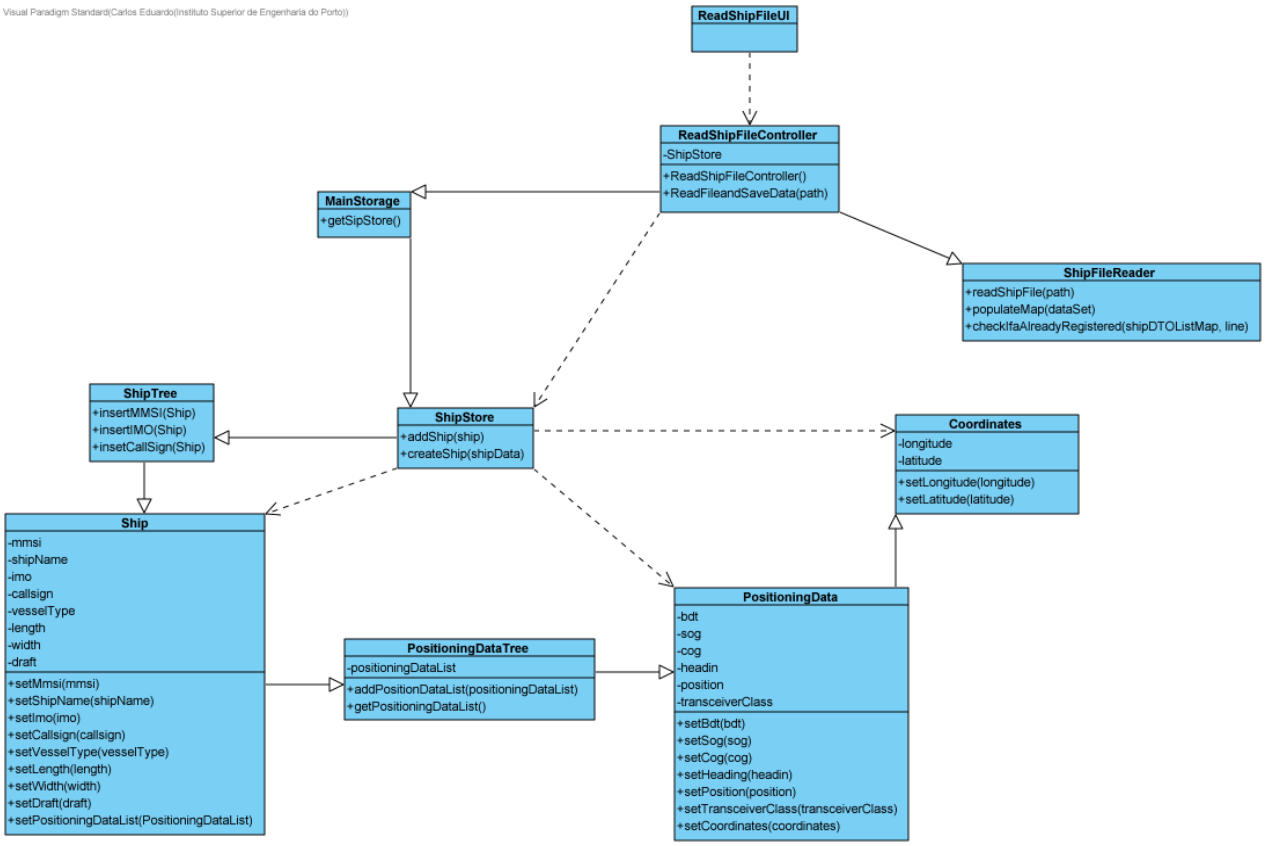
DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

isep Instituto Superior de Engenharia do Porto

Índice

US101

As a traffic manager, I which to import ships from a text file into a BST



Este Diagrama de Classes consta de ambas classes de modelo e classes responsaveis por tratamento de dados. As classes de modelo, PositioningData, Coordinates Ship são a base desta User Story pois constituem os objetos que sao criados a partir da leitura de um ficheiro.

A funcionalidade desta User Story começa na classe do controller em que a classe ShipFileReader é chamada. Como esta classe passa pelas linhas todas presentes no ficheiro de dados ,**O(n)**, e posteriormente armazena estes dados numa lista de Dtos e, enquanto passa pelas linhas do ficheiro infere se o dado Ship ja terá sido inserido nesta lista,**O(n)**, a sua complexidade acaba por ser **O(n^2).**

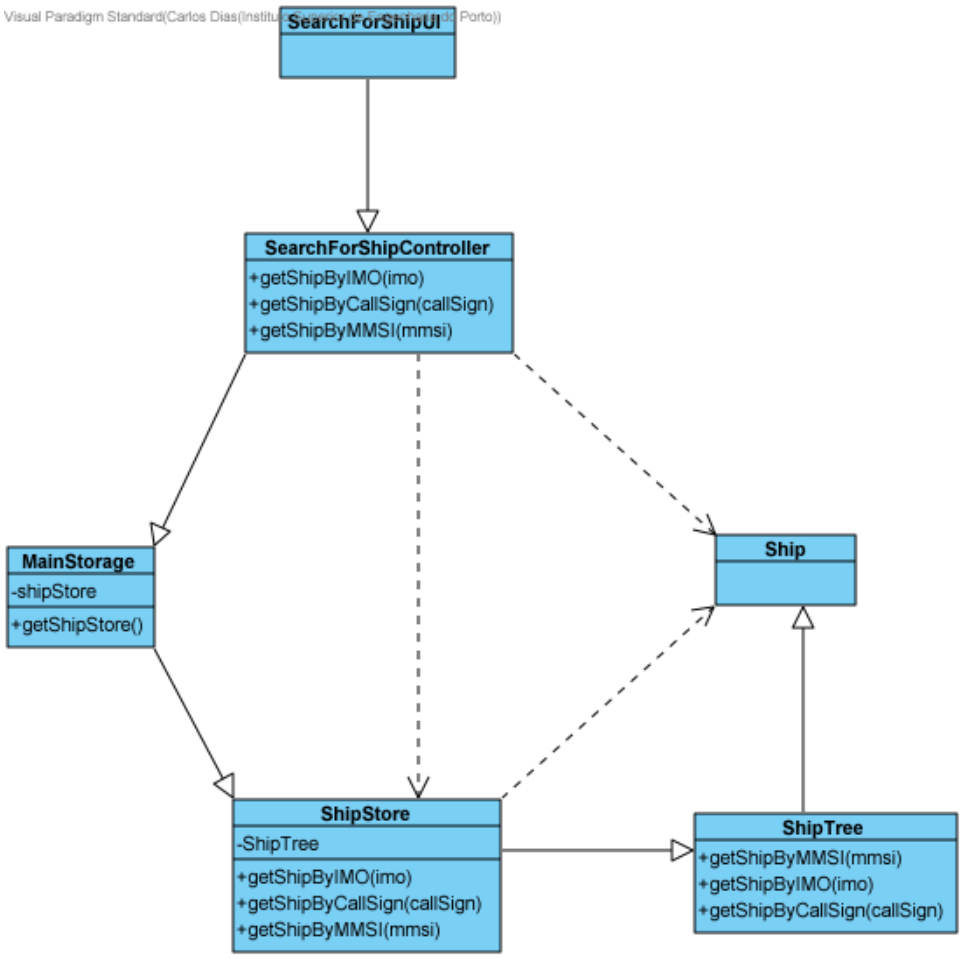
De seguida o controller transforma toda a informação contida nos Dtos e transforma-a em objetos ,**O(n)**, sendo que os objetos de data dynamica são depois introduzidos numa AVL tree na classe PositioningDataTree cuja complexidade de inserção é **O(log n)**, fazendo esta operação de complexidade **O(nlog n);**

Finalmente o controller adiciona todos ships criados a uma nova AVL tree na classe ShipStore cuja complexidade é **O(log n).**

Desta forma podemos averiguar que a complexidade final desta US é **O(n^2).**

US102

As a traffic manager I which to search the details of a ship using any of its codes: MMSI, IMO or Call Sign.



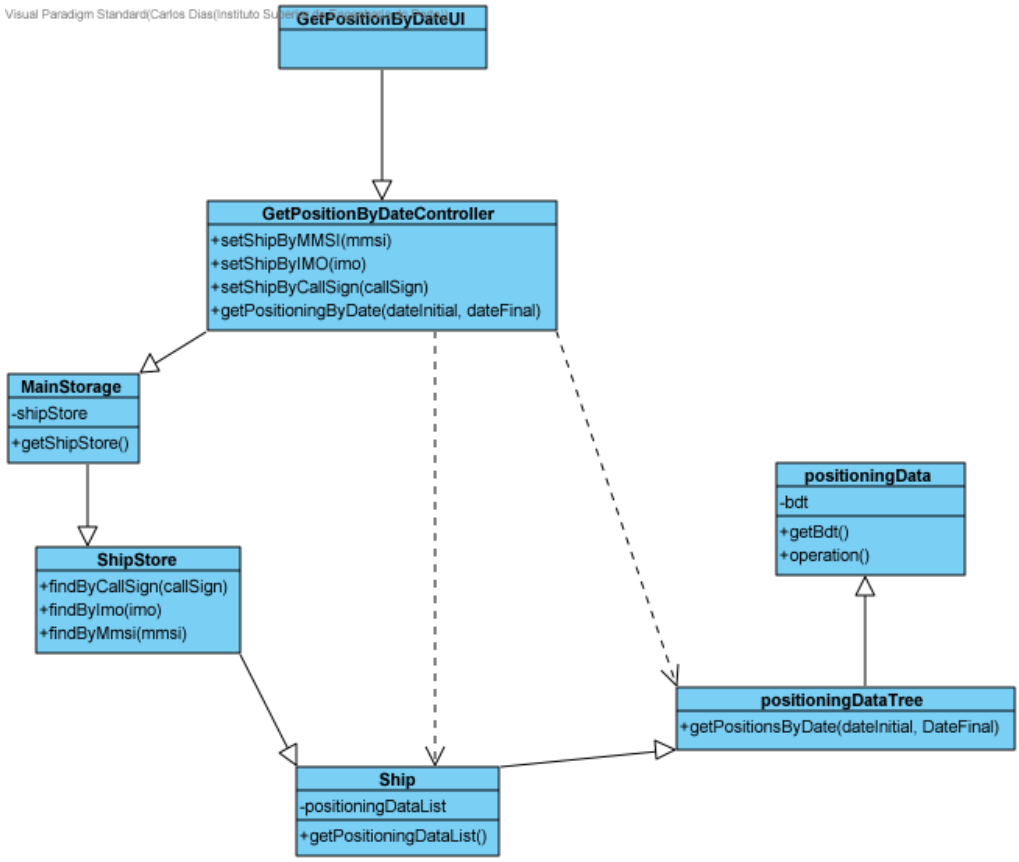
Nesta User Story o controller aceita uma string para procurar um dos navios guardados em ShipStore por um dos seus valores principais.

Como a arvore esta organizada pelo valor MMSI do navio, no caso da procura ser feita pelo valor do IMO ou do Call Sign usas-se 2 hasmaps que guardam os valores dos varios IMO ou Call Sign e do seu repectivo MMSI **O(1)**, apos obter of valor MMSI desejado usa-se o metodo findMMSI que procura pelos ramos da arvore ate encontrar o navio desejado, este metodo tem a complexidade **O(log n)**.

Desta forma esta User Story tem a complexidade O(log n).

US103

As a traffic manager I which to have the positional messages temporally organized and associated with each of the ships



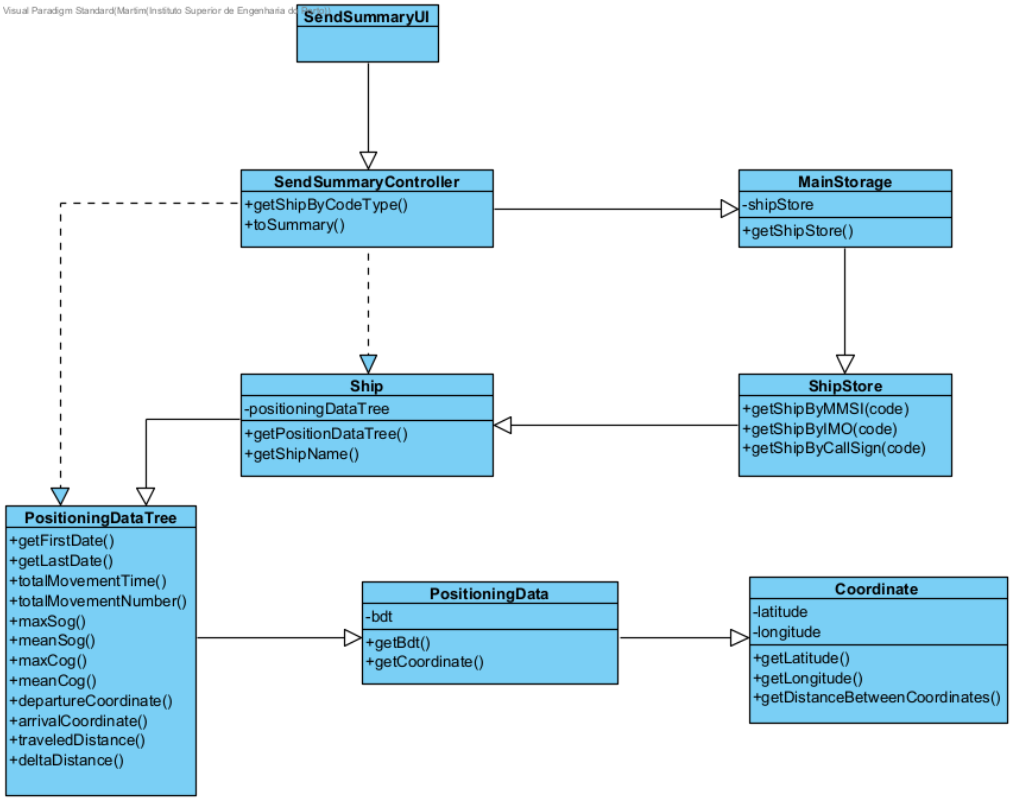
Esta user story começa por procurar por um expecifico navio na ShipStore, esta operação funciona igualmente à User Story 2 logo a sua complexidade será **O(log n).**

Depois de obter o navio desejado o controlled pretende obter todas as posições deste navio num determinado periodo de tempo, este metodo devolve uma nova arvore de posições atravessando este periodo de tempo organizadas, assim, como o pior caso é que as datas englobem a totalidade da arvore principal a complexidade de passagem pela árvore é **O(n)**, e a complexidade de inserção destas datas na nova tree é **O(log n)**, logo a complexidade deste metodo é **O(nlog n).**

Com isto aformamos que a complexidade temporal da US3 é **O(nlog n)**.

US104

As a traffic manager I which to make a Summary of a ship's movements



Este diagrama de classes contêm ambas as classes de modelo e as classes responsaveis pelos dados a ser usados pela User Story. A classe PositioningDataTree é uma das mais importantes da User Story, uma vez que é esta que permite efetuar todos os cálculos e pesquisas necessárias para a realização da User Story.

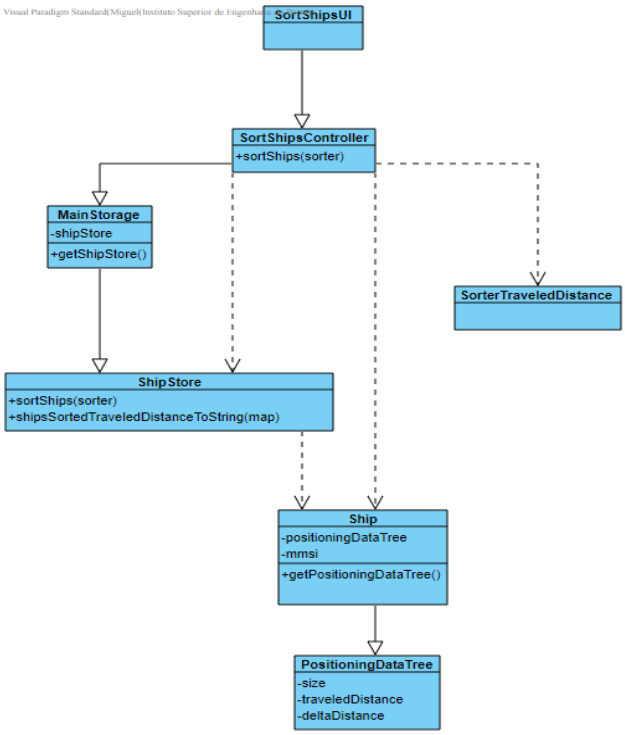
O objetivo desta User Story é mostrar um sumário de um dado navio ao utilizador, sendo esta navio escolhido pelo utilizador através do código MMSI, IMO, ou Call Sign. O sumário deve conter, por exemplo, a data de partida e de chegada do navio, incluindo as suas coordenadas, a distância total viajada pelo navio, a distância entre o ponto de partida e o ponto de chegada, etc.

A usar story começa no controller, num método que permite obter o navio desejado. Para tal, dá-se ao uso de uma complexidade **O(logn)**, uma vez que procura o navio desejado pelos ramos da árvore, usando o método findMMSI, findIMO ou findCallSign, dependendo do input do utilizador. De seguida, avançamos para o método toSummary, que permite obter o sumário completo do navio, devido ao uso do método traveledDistance da classe PositioningDataTree, sabemos que o método toSummary irá ter complexidade **O(n)**, uma vez que terá de passar por todos os membros da árvore para realizar certos cálculos.

Logo podemos concluir que a complexidade desta User Story será **O(n)**, uma vez que o método toSummary terá que percorrer a árvore completa para efetuar o cálculo da distância total viajada.

US105

As a traffic manager I which to list for all ships the MMSI, the total number of movements, Travelled Distance and Delta Distance



Neste diagrama de classes constam ambas as classes de modelos e responsaveis por tratamento de dados. A classe modelo, 'Ship' e 'PositioningDataTree' são as mais importante pois são a base da user story.

O objetivo desta US é ordenar todos os navios pela sua 'Traveled Distance' e 'Number of Movements' para isso é criado um comparator que irá ser responsavel por colocar todos os navio na sua respetiva ordem. Tendo isso em conta, é necessário passar por todos os navios da árvore sendo a sua complexidade **O(n).**

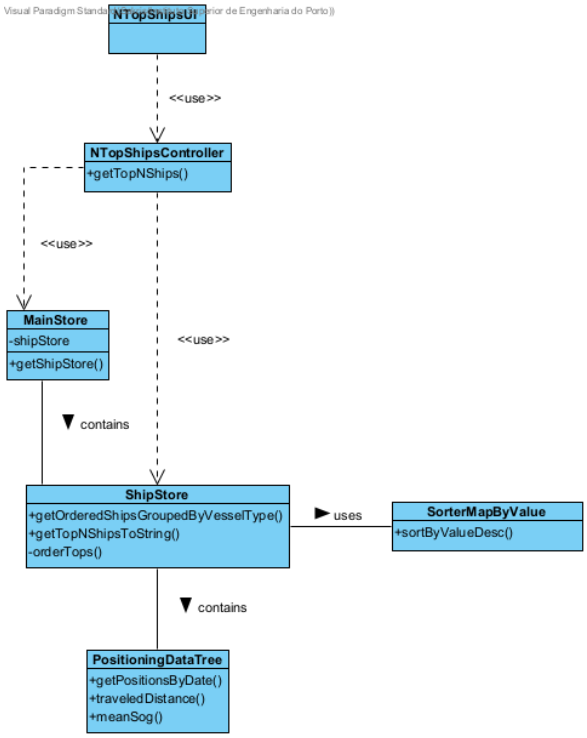
Esta user story começa no controller que pede ao 'ShipStore' por um TreeMap de 'Ships' ordenados. A seleção desta estrutura é essencial porque para além da sua estrutura ser semelhante à da árvore, também permite ao progamador inserir um método de comparação.

Após a obtenção deste mapa é então criada uma lista, com o intuito de apenas mostrar resultados ao utilizador, com todos os navios ordenados e mostrando o seu 'MMSI', 'Traveled Distance', 'Number of Movements' e 'Delta Distance'.

Portanto, em relação à complexidade vão ser inseridos **n** barcos no Treeset sendo a complexidade desta operação **O(n)**. Porém, a operação de adicionar um barco a um TreeSet tem uma complexidade de **O(logn)** e por isso como esta operação será feita n vezes, então a minha complexidade final será de **O(nlogn)**.

US106

Get the top-N ships with the most kilometres travelled and their average speed (MeanSOG).



O diagrama de classe implementa a metedologia Model-View-Controller de OOP onde a camada View é definida pela classe NTopShipsUI, que não foi implementada devido à aplicação não ter interface mas foi incluida na mesma pois pode vir a ser implementada no futuro, a classe NTopShipsController representa a camada Controller, e as restantes classes fazem parte da camada Model.

Para além disso também foi aplicado o padrão Singleton na classe MainStore e o padrão Store na ShipStore, isto para garantir a integridade d código.

As classes base desta user story são a PositioningDataTree, pois esta é que tem maior parte dos dados necessários para realizar esta user story e também é a que vai realizar os calculos necessários, e a ShipStore pois tem os restantes dados necessários e é a classe que vai comunicar com todas as classes necessárias para criar os dados pedidos pela user story.

O objetivo desta user story é devolver tops de ships agrupados por VesselType, sendo que cada VesselType terá um top organizado pela TraveledDistance e outro top que estará organizado por MeanSog. Estes tops apenas correspondem a informação existente entre duas datas fornecidas pelo utilizador.

Sendo assim, para realizar esta user story primeiro vamos iterar por todos os navios na ShipStore, logo **O(n)**, e por cada navio vamos verificar o VesselType e criar um Pair de LinkedHashMap, por cada VesselType diferente que encontrarmos, ambos são colocados num HashMap, cuja key será o VesselType e o Pair será o value, colocar estes dados no HashMap tem uma complexidade de **O(1)**. Para isto será no entanto necessário verificar se o HashMap já tem o VesselType associado a uma key, que

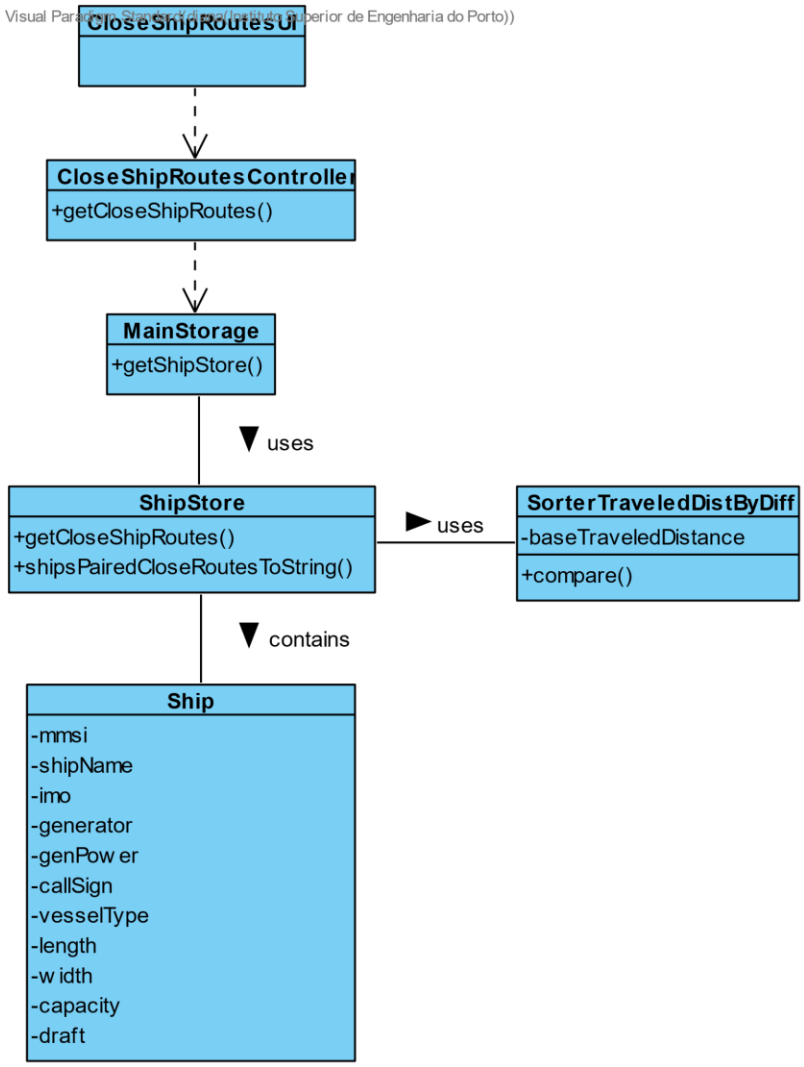
será uma operação de **O(a)**, pois é necessário utilizar o metodo containsKey(), onde “a” é o número total de VesselTypes.

Após isso será gerada uma AVL que vai conter a informação dinamica do barco que se encontra entre as datas fornecidas pelo utilizador, sendo que esta operação tem uma complexidade de **O(nlogn)**. A partir desta AVL será calculado a TraveledDistance e o MeanSog, que terão uma complexidade de **O(n)**. De seguida estes valores são guardados nos respetivos LinkedHashMaps acompanhados do respetivo ship, a complexidade será então de **O(1)**.

Logo podemos concluir que este algoritmo tem de complexidade total **O(nlogn)**.

US107

Return pairs of ships with routes with close departure/arrival coordinates and with different Travelled Distance.



O diagrama de classes definido implementa a metologia Model-View-Controller de OOP em que a classe CloseShipsUI corresponde à camada View (não implementado, uma vez que a aplicação não tem interface, no entanto incluído por motivos de possíveis desenvolviementos futuros) e a classe CloseShipsController representa a camada Controller.

Aplicou-se ainda o padrão Singleton na classe MainStorage e o padrão Store na ShipStore, de modo a garantir a integridade de código. Por fim, a classe SorterTraveledDistance – a classe base desta user story - consiste numa herança da classe Comparator, a usar nas estruturas de dados com ordenação natural.

O objetivo desta US é devolver pares de navios com rotas com coordenadas de partida/chegada próximas (de modo a que ambas não distem mais do que 5 Kms) e com diferentes TraveledDistance, ordenado pelo código MMSI do 1º navio e por ordem decrescente da diferença de TraveledDistance.

Para isso, são feitas todas as combinações possíveis, ignorando as repetições, seguindo uma lógica de matriz:

	Ship1	**Ship2**	**Ship3**
Ship1			
Ship2	(Ship2, Ship1)		
Ship3	(Ship3, Ship1)	(Ship3, Ship2)	

A complexidade deste método de geração de pares consiste em percorrer todos os navios $\frac{n^2}{2}$ vezes, logo **O(n^2).** À medida que os pares são gerados, são inseridos num HashMap, cuja Key é o primeiro navio (ficando já ordenados por MMS, uma vez que são extraídos diretamente da ShipTree por ordem) e o Value é um TreeSet de navios que corresponde a todos os pares possíveis do primeiro navio, cuja ordenação é definida por um Comparator dedicado. Neste contexto, a inserção no HashMap é de complexidade **O(1)**, enquanto que a inserção no TreeSet é de **O($\log_n$)**. No total, o algoritmo de geração de pares e ordenação de acordo com os critérios estabelecidos é de **O(n^2)**.